e-Science Experiences:  
Software Engineering Practice and the EU DataGrid

Lee Momtahan and Andrew Martin
Oxford University Computing Laboratory

Abstract

The conduct of collaborative scientific study mediated by the internet — e-Science — is giving rise to a new type of large distributed software project. This paper reports initial experiences of one such project: the European DataGrid. We record some observations about the intended lifecycle and process, compared with actual practice. The paper explores the applicability of current software development practices from the academic, commercial, and open source sectors in the context of such Grid projects.

1 Introduction

The increasing interconnectedness of the world has given rise to the Grid computing paradigm [4]. In the academic world, a major driver for Grid development is collaborative science mediated through technology — e-Science [8]. Whilst scientists of many disciplines have been using computing technology for decades (almost pre-dating computing science itself), e-Science projects present fresh challenges for a number of reasons — for example, their scale.

The EU DataGrid is a €9.8m project, funded through the European Union Framework IV [14]. It has 21 partners in 15 countries, with a total of 200 people directly involved over a period of three years, starting in 2001. The objectives of the project are to support advanced scientific exploration within a Grid infrastructure, offering capabilities for intensive computation and analysis of shared large-scaled databases, from hundreds of Terabytes to PetaBytes, across widely distributed scientific communities. Such requirements are emerging in many scientific disciplines, including physics, bioinformatics, and earth sciences.

The authors are part of a project funded by UK research councils PPARC (Particle Physics) and EPSRC (Engineering and Physical Sciences) intending to bring perspectives from Computing Science and Software Engineering to the DataGrid initiative. In this paper, we record some initial observations about the intended lifecycle and process compared with actual practice and explore the extent to which familiar Software Engineering models can be applied in this new context, and the extent to which novel models are needed.

Some of the concerns are of course determined by social and political contexts. We expect to be able to draw common themes which will apply to other e-Science projects (and similar distributed collaborative projects), recognising that constraints of timing, funding, and partner involvement may have a large effect.

Section 2 describes relevant features of the EU DataGrid. Section 3 records some problems that appear to be arising. In Section 4, the lifecycle model and context of the DataGrid are compared with other established Software Engineering contexts. The paper ends with some conclusions.

2 The EU DataGrid

Grid software implements the tools, middleware, and services necessary to:

- build (possibly application-specific) application frameworks potentially involving huge amounts of data, compute power and distribution;
- provide secure, managed, uniform access to such resource;
- facilitate collaboration, and remote access to data and scientific instruments; and
- manage such facilities as a persistent service.

Grids are frequently described as promoting virtual organisations which allow stakeholders from various real organisations to work together.

The EU DataGrid project aims to develop, implement and exploit a large-scale data and CPU-oriented computational Grid. It is one of the largest such projects to date; it is comparable with the US Particle Physics Data Grid (ppdg.net). The Grid will allow large distributed datasets and CPU-intensive scientific computing models to be combined using a hugely distributed testbed.
The project is based around the Globus Toolkit, [4, chapter 11]. Globus provides the basic infrastructure for the creation of Grids — chiefly the lower levels of the middleware which provide services such as user authentication, communication, basic scheduling of CPU jobs, data transfer, etc.

The project aims to build on this basis by developing higher-level services (‘Grid Collective Services’) for workload management and data management, and also fabric and mass storage management, and monitoring. Development of these entails work on Grid-enabling database engines for the storage of meta-data, management of data replication and cataloging, optimal scheduling of multiple resources, and so on.

This work is informed by the requirements of a number of application areas: Particle Physics, Earth Observation, and Bioinformatics. Software applications for these areas are under construction, and their requirements are intended to drive the development of the middleware. In the next few years these communities will need to access and process large quantities of data which exceed the present quantities by at least an order of magnitude. Present systems will not scale or will be unaffordable for the participating institutions [9, section 3].

The various activities described are grouped into a number of workpackages, each of which involves a subset of the 21 distributed partners. Figure 1 illustrates the broad relationships between these workpackages.

A further workpackage covers project management, largely undertaken at CERN. Project management and technical boards are widely-drawn, to provide oversight for the project, and an Architecture Task Force draws together members of each of the workpackages to define the project’s overall software architecture.

Project members are widely distributed geographically. Most communication happens by phone and email, with occasional face-to-face meetings of work packages and project boards. The whole project holds an biannual working meeting/conference, with extensive discussions within and between workpackages.

3 Challenges

In this section we discuss the some of the Software Engineering challenges which appear to characterise the EU DataGrid. These are by no means unique to the project, but their particular combination appears to characterise many e-Science projects.

3.1 Volatility of requirements

In common with many Software Engineering projects one of the challenges that faces the EU-DataGrid is a difficulty in capturing requirements.

The various application groups (Particle Physicists, Bioinformaticists and Earth Observationists) each have different but overlapping requirements. For example, all are interested in large data-sets, but for the particle physicists these are an order of magnitude larger (petabytes) than for the other application areas. A significant project activity is to determine and reconcile these requirements, identifying commonality wherever possible. This is an ongoing process, while the middleware is being defined and constructed.

As importantly, because Grid computing is a novel paradigm, and a Grid with so many diverse partners has never been built before, many features have a high degree of novelty. It is difficult to see what value a feature may have until it is used for real. Technical features which may sound
ideal might turn out to be little used; the seemingly mundane facilities may turn out to be the ‘killer applications’. This kind of evolutionary design has certainly been the case with the World-Wide Web. Many of its modes of use could not be foreseen (at least not with clarity) when the web was in its infancy. The success of the web must be in large part due to the appropriateness, genericity and simplicity of its core components.

Conversely, it is only when a working system is delivered and used in context that some of the shortcomings can be seen. This is foreseen in the project plan and reflected by the evolutionary lifecycle model used [5, section 9.1].

The EU-DataGrid software uses many off the shelf components, but some of these are themselves volatile. The main example here is the Globus Toolkit, which provides the fundamental infrastructure and framework for the project’s software. When the EU-DataGrid project started Globus version 1 was current. Version 3 will be the main supported release before the end of the project. The toolkit has been very substantially restrucutred through these different releases.

The desire of the EU-DataGrid to take advantage of the latest developments in Globus and other developments in the rapidly progressing e-Science arena is another source of volatility [11, section 4.4]. This is necessarily a challenge for a project which seeks both to undertake leading-edge research in practical distributed computing, and also aims to produce a production Grid environment which can be used for scientific applications: the latter requires stability, the former (and retaining technical interest of project staff) suggests a need to keep the project current with latest versions.

### 3.2 Geographical Separation and Communication

It is well known that communication between the various members of a large software engineering project can become the limiting factor to progress [3, chapter 2]. This difficulty is compounded by the geographical separation of project members across 21 partners [10], [15, section 2.1].

Some of the application groups are well used to this kind of distributed collaboration—it is the norm in particle physics, where the very expensive equipment required tends to concentrate practical work on a few experimental sites. Scientists visit the sites, and interact through electronic means quite routinely. However, with the development of the DataGrid, the context calls for a much more closely-coupled collaboration (whether it should is a different matter; see below), producing software with detailed and complex interfaces.

The software engineers involved may be used to doing research together, but producing software together is a different matter. The distance between project groups magnifies the communication overhead and this can result in misunderstandings and tensions between various groups. Some project members can feel left out. The familiar problem of project documentation quickly becoming out of date (due to changes in requirements and design) is also magnified.

### 3.3 System Decomposition

Given the challenges already described, initial decisions on how to decompose the design and implementation into subsystems have a huge impact on the project’s progress.

In an industrial software engineering context, traditional good practice would entail breaking the problem into functional subsystems and allocating each to a team which sits together in one location (or at least a very small number of locations). Even when components are outsourced, it would be usual to define their interfaces quite closely. It is widely held that the decomposition of the task can make or break a project. Brookes [3, chapter 4] argues that only a small technical elite can ensure conceptual integrity.

In a project like the DataGrid, the workpackage breakdown is not driven by purely functional or technical concerns. Not only do individual sites have particular expertise or interest in particular areas, it is also in the nature of academic research to seek to build as large a group as possible, for the sake of reputation and receipt of overheads.

The project plan defines five work packages, each building part of the Grid middleware:

- Grid Workload Management
- Grid Data Management
- Grid Monitoring Services
- Fabric Management
- Mass Storage Management

Each work package begins by gathering requirements and designing part of the system and interfaces for other work packages to use. Although an Architecture Task Force (with representatives from each work package) has been defined to assure the overall design and technical consistency of the software, the ATF’s architecture document took as input the architecture specifications from the work packages rather than the other way around [5].

The composition of the designs of each of the work packages was not designed at a high level of abstraction [6, section 9.3]. With the benefit of hindsight, it is not surprising that the composition does not factor out easily into a high level abstraction. This may tend to increase the communication challenges facing the project and lead to some inevitable duplication of work. End-to-end requirements, which span multiple middleware work packages are hard to assure [6,
The formation of the project’s Security Group after the project’s inception helps to highlight this point – the need for this group was not predicted at the start of the project, but later it became clear that the security concern cross-cut the various workpackages.

3.4 Project Processes and Authority

Early in the project, a number of procedures and processes were documented in the Project Quality Plan [15]. This and other project management documents set out best practice for the project, based largely on good industrial practice.

Unfortunately, the mere definition of such processes does not bring them about — individuals typically prefer their own informal working processes. When such activities are described in an industrial context, the project manager and technical leaders have considerable leverage in making sure they are followed. This does not carry over well into the academic context, especially because of the widespread distribution of staff in this case. Moreover, the following (or not) of procedures is very largely a matter of cultures and organisations (see the discussion of CMM at Section 4.2). Even if research officers are aware of a quality plan, it appears to be hard to persuade them to take account of it. For example the Project Quality Plan in section 2.2 describes project deliverable 12.3 as containing a Project Development Plan and specifies its contents in section 5.4, but the deliverable 12.3 clearly does not meet this specification [15], [16].

Of course, academe has its own quality control procedures and customs, largely based on peer review of proposals, papers, and reports. We may ask in passing how to define best practice for academic development of software, and whether the processes traditionally applied to papers can be applied to software.

3.5 Project planning and tracking

Although different models are under discussion, most software development proceeds with a curved resource profile: in early phases, few personnel are committed to the project as its specification, design and decomposition are considered. More resources are added through development and testing phases, probably ramping back down as final delivery nears. This is illustrated in Figure 2a. (An interesting exception appears to be Agile software methodologies such as eXtreme Programming, which might employ constant resources).

By contrast, research projects, especially publicly-funded ones such as this, tend to have a very flat resource profile (Figure 2b), although this hides the fact that some preliminary, unfunded work will have been necessary in proposing the project. This flat profile arises both because of the funding culture external to the institutions involved and also because, in order to retain staff they must allocate individuals to projects in a sequential—perhaps simplistic—manner.

We have observed that tracking of actual costs against planned is good, however tracking of task completion against plan has been poor and milestones could be defined more crisply. For example the exit criteria for an ‘iteration’ seems to be the completion of a document detailing the problems that were found in testing. This is highly unconventional [12].

Also, the human resources required to implement the middleware work packages were set out in the project plan at the inception of the project, but the rationale behind these estimates is not given. These estimates precede the initial requirements gathering and design tasks executed by these work packages.

3.6 Morale, Attrition, Culture

Academic projects such as the EU DataGrid appear to be characterised by low pay and low attrition rates: many project staff could presumably command higher salaries in the private sector, yet having chosen to work in academe are unlikely to leave the project. This stands in stark contrast to a competitive business environment, where staff retention is often a problem.

From anecdotal evidence we have formed the impression that some of the people on the project feel that they have responsibility that is not commensurate with their authority and this is causing frustration — morale is not especially high. It is therefore all the more surprising that attrition on the project is virtually zero. This is fortunate, as a high
rate of attrition combined with the documentation difficulties we outlined earlier could have been truly disastrous for the project.

One exception is the Project Architect [5] who was hired from industry at the start of the project, but left shortly after. This post was not refilled. However a ‘technical coordinator’ is overseeing the project. The term ‘Project Architect’ now seems almost taboo among project staff, which seems indicative of the day-to-day project culture and contrasts with an industrial project.

4 Software Production Practice

We may summarise the emergent software context implied by the preceding section with a number of key points. The combination of these highlights the ways in which the DataGrid project is unusual compared to classically-defined software engineering projects:

• enormous geographical distribution — over 20 sites participating in the collaboration;
• absence of an authoritative/authoritarian management structure for the project;
• requirements from a number of different stakeholders with different perspectives;
• evolving requirements and design due to novelty of software; and
• ‘square pulse’ funding/human resource allocation.

4.1 Communication Overhead

We recall Brookes’s [3] observation that the time taken to complete a software project is not monotone decreasing with the number of individuals, due to the greater-than-linear communication overhead. With a classical development process this communication overhead becomes dominant much sooner with the geographical separation of project members on this project. The graph at Figure 3 underlines the observation that at some point more sites and project individuals makes the rate of production decrease.

The discussion in the Challenges section of this paper seems to indicate a severe communication bottleneck and this would suggest that the project is towards the right of the graph. We have not found any evidence to suggest otherwise.

4.2 SEI Capability Maturity Model

The Capability Maturity Model for Software [7, CMM], has become widely accepted as a means to describe organisations and their software development processes. It ranges from level 1: ‘initial’ (ad hoc, unplanned, few defined processes) through ‘repeatable’, ‘defined’ (processes documented, standardized, organization-wide), ’managed’ (adding quantitative recording understanding) to level 5, ‘optimizing’ (with continuous process improvement).

For the DataGrid, although documented procedures do exist, there is no authority to enforce that procedures are followed by all project members. Therefore it is hard to avoid the conclusion that this project is at maturity level 1 (‘initial’).

We may sensibly ask whether this is necessarily the case for projects undertaken in this way. In the past, academic projects have tended to involve modest software development, with little in the way of software integration, and a level 1 process may be suitable. Of course it is disturbing to ask how many project conclusions — especially in the medical field — have been formed erroneously thanks to poor design of the software used in data processing.

In other areas of scientific endeavour (for example, the production of academic papers, or the conduct of experiments) the maturity level is much higher (some would suggest that it is now too ‘managed’, though undoubtably those institutes working towards a ‘optimizing’ level are those which regularly achieve high and stable levels of funding). This is achieved without prejudice to the development of new methodologies, ideas, and techniques. It seems reasonable to argue that scientific code production should likewise strive towards a better process.

4.3 Academic Software Engineering Practice

Unlike industry, where each company has its own software engineering process, tailored to its niche within the marketplace, there are no papers or body of knowledge defining the software process to be used by academics in general. We observe that most projects define their own process based on the software engineering research that has been informed by industrial practice, or they proceed on an ad hoc basis. A disadvantage with this approach is that lessons learnt on one project are not always carried over to future projects. Also, this approach does not seem to scale well to the size and complexity of projects like the European DataGrid.
4.4 Open Source Models

The term *open source* narrowly refers to a kind of licence and/or method for distributing software. Source code for the European DataGrid project is distributed under such a licence: it is freely available for download and use.

However, the term has gained a richer meaning, in referring to the way that open source software is developed for example with the Linux and Apache projects. This is a software development methodology in its own right, albeit perhaps an uneven one which is applied differently in different projects.

The open source method can be described as an unusually rapid and iterative form of Boehm’s spiral model with loops that might take as little as hours to complete. Open source projects attempt to ship out minimally working prototypes at the earliest possible time and by doing so receive feedback on their features and designs very early in the overall development process [2]. Open source can side-step the greater-than-linear communication overhead recognised by Brooks partly because the vast majority of individuals contributing to Open Source projects are raising problems. The quick turn-around-time made possible by the rapid spiral model reduces duplicate defects, thus the testing process can be efficiently dispatched to multiple individuals and sites. Mockus et al. [1] hypothesise that successful Open Source projects have a small core of approximately 10-15 controlling the code base and adding most of the new functionality, a group larger by an order of magnitude repairing problems, and a yet larger group (by another order of magnitude) reporting problems.

The issues listed at the start of this section (page 5) seem to suggest a good match with the accepted features of an open source software project. Eric Raymond [13, ‘The Magic Cauldon’] claims the following discriminators push towards open source:

- reliability/stability/scalability are critical
- correctness of design and implementation cannot readily be verified by means other than independent peer review
- the software is critical to the user’s control of his/her business
- the software establishes or enables a common computing and communications infrastructure
- key methods (or functional equivalents of them) are apart of common engineering knowledge

These discriminators seem to apply to the European DataGrid and other Grid computing software. Perhaps some of the key methods referred to in the last point are common only within the academic arena.

Continuing the parallels that Grid computing often tries to draw with the World Wide Web, we note that the Apache web server, which is developed under the open source paradigm accounts for the majority of web servers on the Net (www.netcraft.com). Also, much of the infrastructure of the Internet is developed in this way, for example, bind (DNS), the Linux kernel, sendmail, and Perl.

5 Conclusion

We have described some of the apparent software engineering challenges in the EU DataGrid. Despite early good intentions [15] and significant progress [11], problems remain which may reasonably be expected to be generic in the e-Science arena. The challenges for such endeavours are not merely in the design of experiments and conduct of science in this way, nor merely in the design and development of protocols and software to facilitate that science, but also in the processes to be employed in achieving those goals.

The difficulty of decomposing the system and partitioning the workload amongst project teams from the outset seems to be the key to the project’s experiences to date. Our project is hoping to assist in a rational reconstruction of elements of the system, by way of clean abstractions, with a view to assisting development where it might be likely to stall.

We tentatively conclude the practice of software engineering in an e-Science context is substantially different to practice within an industrial context and has much in common with the Open Source model. Most software engineering wisdom has arisen in the industrial context and the evidence on which this is based appears ‘skewed’ when applied to academe. It may be that weight should be given to practices used in the Open Source community when designing academic projects. Current academic practices are struggling to cope with the scale and complexity of e-Science and further research is required to find methods suitable to academies traditionally productive style.

The Globus software is instructive in this regard. It is distributed in open source, but clearly owned by Argonne National Laboratories in the USA. Whilst it is not strongly in the second sense of ‘open source’ above, the developers are keenly making available interim releases and patches, in the style of an open source project. The EU DataGrid has found many bugs for them in the Globus Toolkit 2 beta release, and presumably this arrangement significantly helped Globus.

Development in an open source style may or may not help academic institutions to reach CMM level 2 in e-Science projects. In other fields of scientific endeavour, however, publication and review are the key to quality and process improvement. It may be that the publication of software to an open repository, and its review through test execution and integration is the way forward.
References


Acknowledgements

The authors’ project is funded through UK research councils PPARC and EPSRC as grant GR/R74284/01.

We thank members of the EU DataGrid project for discussions which have helped us to form the impressions reported in this paper. Figure 1 is taken from DataGrid documentation.

‘Capability Maturity Model’, and CMM are trademarks of the Software Engineering Institute, Carnegie Mellon.